Processorless Smart Sensors with Distributed Intelligence

Eric Benoit, Chotin Eric, Laurent Foulloy

Proc. of the 14th IMEKO World Congress, vol. V, Tampere, Finland, June 1997, pp. 60--65.

PROCESSORLESS SMART SENSORS WITH DISTRIBUTED INTELLIGENCE

Eric Benoit, Eric Chotin and Laurent Foulloy
Laboratoire d'Automatique et de MicroInformatique Industrielle,
Université de Savoie, Annecy, France

Abstract: In the proposed approach, smart sensors own the definition of the software functionalities but are no longer able to execute them locally. Thanks to the network, these software functionalities are sent to a smart sensor, to a smart actuator or to a common resource that has computation facilities. Due to the wide range of possible processing units, the exchange of software functionalities takes place at the source level. A dedicated language, has been especially developed to design smart sensor integrating fuzzy functionalities. In order to perform the remote execution of the code, each unit with computing capabilities owns a compiler. Processorless smart sensors send the source code to a computing unit which will execute the code and send back the results. This paper presents the concept of intelligent cells. Then processorless smart cells are introduced. The main topics of the dedicated language are presented. A hardware solution for the implementation is detailed. An illustrative example is provided.

Keywords: intelligent sensor, distributed intelligence, field bus

1. INTRODUCTION

Since the eighties, the concept of smart cells communicating over a field bus network in order to drive an industrial process has been developed [1][2][3][4]. It is commonly admitted that smart cells (i.e. smart sensors or smart actuators) include functionalities such as measurement, communication, configuration and validation. Generally several functions are performed by hardware like basic measurement or network interfacing while the others are performed by software like signal processing or diagnostic. Usually, smart cells include a processing unit whose computing ability allows them to perform all software functions. However, including a processor into each smart cell is sometimes expensive especially when the cost of the processor is high compared to the cost of the sensing unit.

Our approach is to consider that an intelligent cell does not need to own the ability to compute its own functionalities if another cell can perform them instead of it over the network. In fact, an intelligent cell only needs to owns its functionality definitions assuming that another one can perform them. In order to implement this concept, we propose to code the intelligent functionalities of each cell into a source code written in a dedicated language called PLICAS. This code can contain configuration parameters and definition processes used to implement intelligence. This source code can then be transmitted to another cell to be executed.

The aim of this approach is to increase the interchangeability i.e. the possibility to replace a cell by another one, and the interoperability i.e. the possibility for cells to communicate with other cells [5].

2. INTELLIGENT CELLS

2.1 Functional description

An intelligent cell is a material unit which owns functionalities in addition to sensor ones or actuator ones. Common functionalities can be communication over a network, validation, diagnostic, measurement, signal processing, control.

In order to implement such cells, we will consider two distinct levels in the functional

description of a cell, each one being implemented separately: The first level is the general behavioural description. It consists of a state machine describing the different functionalities to be executed by the cell according the occurring events. This state machine is implemented locally. The second level corresponds to the functionalities which are available in the cell, like signal processing, diagnosis. These functionalities are performed by software and are implemented as source codes containing configuration parameters and computing routines. This code is stored in a local memory and requires a processing unit to be executed. This processing unit can be local or somewhere else over the network.

In a general way, the set of offered functionalities can be considered as a set of one or more services among the following :

Communication, including local communication and communication over a network.

Validation, including functional validation, technological validation, historic and diagnosis. The functional validation allows cells to get information about the correctness of the produced values. With technological validation, cells can verify the correctness of the hardware. The historic service is needed for validation and diagnosis. Diagnosis allows cells to find the origin of a problem.

Configuration, including technological and functional configuration. The technological configuration is usually performed before the installation of the cell in the application. This configuration is relevant to all parameters which depend on the general use of the cell. The functional configuration is the in line configuration used to adapt the cell to a specified application, or to adapt it to a modification of the application context.

Measurement, including basic sensing, signal processing, correction.

The method chosen to implement these functionalities is to use a dedicated language. This language called PLICAS (Prototype of Language for linguistic Actuators and Sensors) is a language initially dedicated to fuzzy control applications [6][7].

2.2 Processorless cell

Generally, the concept of intelligence is associated with the possession of a dedicated processing unit. We propose in our approach, to share the processing units over the network, and then to reformulate the intelligence concept as the ability for a cell to use a processing unit that could be its own or a delocalized one. Thus in the case of a processorless cell, the intelligence consists of a behavioural description of the cell that allows the use of the network resources to carry out the various cell functionalities. The PLICAS code will be sent by the state machine level to a computing resource over the network, and the execution results will be sent back.

The interest of storing the functionality codes locally in the processorless cell, is to allow easy changes in the process assignment on computing cells. This is in opposition with a scheme where the code would be resident on a pre-defined computing cell.

The structural description of a cell is mapped from the functional description: the general behaviour of a processorless cell is implemented using a FPGA carrying out the management system of the cell. The choice of a FPGA illustrates the weak processing resource such a cell needs. The functionalities of the cell are implemented as PLICAS code stored in a read only memory. These codes are transmitted through the network by the FPGA state machine, using a network interface implementing the data-link and the physical layers of the fieldbus protocol. When the cell used must be interfaced with the physical world, it also need transducers, conditioners and analog-to-numeric converters or numeric-to-analog converters.

2.3 Computing cell

The role of a computing cell is to carry out its own treatments as well as the treatments submitted by other cells. To do this, the functional description of the cell is similar to that of a processorless cell, except that the general behavioural description is implemented as a multitasking system, in order to execute simultaneously the local and the delocalized tasks.

These cells include computing resources like microprocessor or dedicated processors like DSP (Digital Signal Processor) or FFT circuits (Fast Fourier Transform). In this case, the cell can compute a piece or all of its functionalities. It can also use its computing resource to compute the functionalities of another cell.

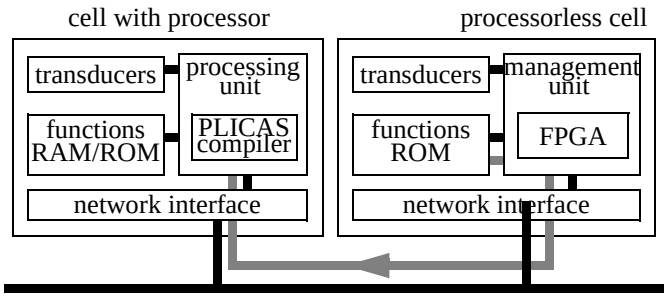

Figure 1. Structural description of intelligent cells

3. PLICAS ENVIRONMENT

3.1 The main shell

In order to perform functionalities coded in PLICAS, each computing cell contains an operating system which allows the management of PLICAS code, the execution of the code and the communication with other cells. This operating system is based on a usual multitasking operating system, a compiler of the PLICAS language and an executive manager.

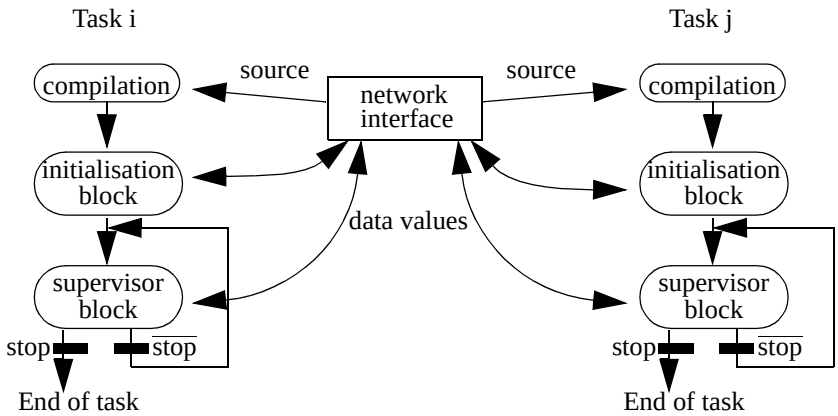

Figure 2. Example of two simultaneous tasks created to manage different PLICAS codes

In the initial state, a main task is running on the cell, waiting for PLICAS source codes. For each received source code, a new task is launched. These codes can be local or coming from the

network. The new task compiles the code by creating a structure in memory which includes the sequence of all its actions. Then the task executes the actions by running the executive manager.

After the compilation phase, the executive manager performs actions coded in the structure. A PLICAS source code is a set of blocks of actions. It needs to own two special blocks called an “initialization block” and a “supervisor block”. The initialization block is executed once at the beginning. After the initialization block is executed, the supervisor block is executed periodically. Each of these blocks can own network instructions to get or send values over the network. This is especially the case when the code is issued from a processorless cell which is waiting for a result. After each execution of the supervisor block a variable called “stop” is tested. If this variable is set then the task ends.

3.2 The language

The PLICAS language owns basic functionalities of sequential programming. In order to minimize the size of the compiler code, the language uses only global variables and does not allow new function definitions. It’s a language created to manage the sequencement of predefined functions. It includes basic arithmetic and logical functions needed to perform this sequencement. Specialized functions like FFT or rule based controller can be added to the language during a technological configuration. Added functions are then used as primitive functions of the language. Each networked variable is managed by a number. A program can then get a variable on the network or diffuse another one. The function “get(var_number)” can perform two actions depending on the localization of the value of the variable. If this value is already present, the function simply returns it. If the value is not present, the function sends a request for this variable over the network.

```
declarations
    double X,Y,Z,Xn,Yn,Zn;
    double L,a,b;

block light_measure
    Xn=get(3);
    Yn=get(4);
    Zn=get(5);

block measure
    X=get(0)/Xn;
    if X>0.008856 then
        X=pow(X,1/3);
    else
        X=7.787*X+(16/116);
Y=get(1)/Yn;
    if Y>0.008856 then
        Y=pow(Y,1/3);
    else
        Y=7.787*Y+(16/116);
Z=get(2)/Zn;
    if Z>0.008856 then
        Z=pow(Z,1/3);
    else
        Z=7.787*Z+(16/116);

block initialization
    execute(light_measure);

block supervisor
    if exist(4) then
        execute(light_measure);
    execute(measure);
    L=116*Y-16
    a=500*(X-Y);
    b=500*(Y-Z);
    diffuse(L,6);
    diffuse(a,7);
    diffuse(b,8);
```

Figure 3. Example of a PLICAS code

4. APPLICATION

This concept is applied on a VAN fieldbus (Vehicle Area Network). A computing unit based on a PC104 is connected to this network. On this unit the multitask operating system RTC is implemented in order to compute simultaneously 16 source codes.

A processorless cell is implemented as a colour sensor, owning a diagnostic functionality. In normal mode, the sensor performs the colour measurement based on three photometric transducers which return three basic measurements X,Y and Z. In most applications, colour measurement is expressed in the **Lab** coordinates instead of the **XYZ** ones. In this way, usual colour sensors are composed with a perceptive head measuring X,Y,Z and a deported calculator performing the mapping between these two colorimetric coordinates :

$$L = 116 f\left(\frac{Y}{Y_n}\right) - 16$$

$$a = 500\left(f\left(\frac{X}{X_n}\right) - f\left(\frac{Y}{Y_n}\right)\right) \qquad where \qquad f(d) = d^{\frac{1}{3}} \qquad when \qquad d > 0.008856$$

$$f(d) = 7.787(d) + \frac{16}{116} \qquad when \qquad d \le 0.008856$$

$$b = 200\left(f\left(\frac{Y}{Y_n}\right) - f\left(\frac{Z}{Z_n}\right)\right)$$

In our application, the perceptive head is included in a processorless cell. During initialization, the cell performs the Xn,Yn,Zn value measurements, which are characteristic values of the light source integrated in the perceptive head. Then these values are sent to the computing cell together with the PLICAS code of the Lab transformation.

Once initialization is done, the processorless cell enters into normal mode consisting of cyclic measurement of the X,Y,Z values, and their transmission to the computing cell.

The diagnostic functionality of the processorless cell is the following : when the characteristic values of the light source change, the new values of Xn,Yn,Zn are sent to the computing cell. This is done by the cyclic measurement of the most characteristic value, Yn, and its comparison with the previous memorized value.

The state machine implementing the general behavioural description of the processorless cell is very simple, even with the network management. This is due to the VAN network functionalities which allow a very convenient mechanism for the response to a network request. This mechanism is call “reply request with immediate reply” or “in frame reply”. It consists of including the response in the request frame, and is completely managed by the data link layer network interface component (29C461A by MHS). The cyclic transmission of the X,Y,Z values is done with this scheme. Thus it only requires storing the responding value together with the frame request identifier in the RAM of the network interface component.

The general state machine of the processorless cell is shown in figure 4.

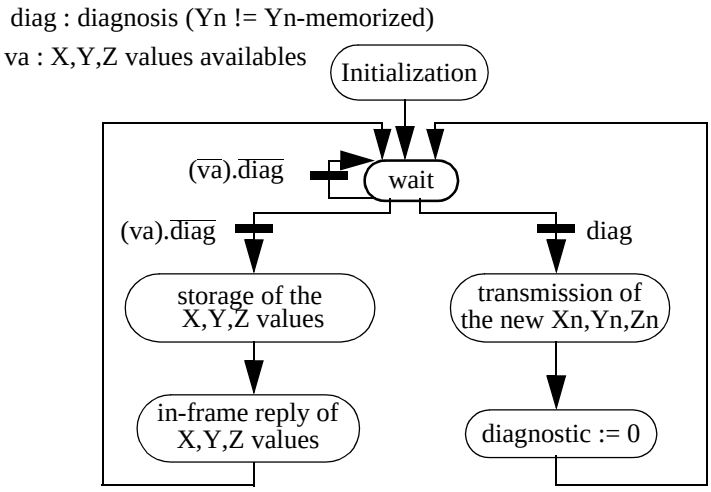

Figure 4. State machine of the processorless sensor

Other data calculation or diagnosis function can be implemented in this sensor. For example, the sensor can verify if the measured colour is included in the set of real colours. If not, it can

conclude that one or more of its transducers is out of order. In another example, the sensor can calculate a fuzzy description of the colour. The way to add these functionalities is to implement the PLICAS source code into the memory of the processorless module, making sure that another cell can perform these functionalities.

5. CONCLUSION

The proposed approach shows the advantage of separating the concept of intelligent process and processing resource. Actually, the confusion between these two concepts implies making considerable changes on an intelligent cell when a new functionality is added on to it. The processorless approach allows the increase in cell functionnalities without any hardware modification. A second advantage of this approach is to improve the interchangeabilty of cells.

REFERENCES

[1] Siebert M., Thomesse J.P., Interworking of Fielddevices, in: *Proc. of the 2nd IFAC Symposium on Intelligent Components and Instruments for Control Applications, SICICA 94*, (Budapest june 94), Hungary, 1994,p. 98-103.

[2] Yagsu T., Support system to construct distributed communication networks - Implementation, in: *Proc. of the 1st European Congress on Fuzzy and Intelligent Technologies, EUFIT 93*, (Aachen Sept. 93), Germany, 1993, p. 532-536.

[3] Iyengar S.S., Kayshyap R.L., Madan R.N., Distributed sensor networks - introduction to the special section, IEEE Trans. in: *Systems, Man, and Cybernetics*, Vol. 21, No 5, Sept-Oct. 1991, p. 1027-1031.

[4] Navaro J.L., Benet G., Albertos P.,Intelligent Distributed Control : A System Architecture, in: *International Conference on Fault Diagnosis*, (Toulouse 5-7. April 1993), France, 1993, pp 127-131.

[5] Staroswiecki M., Bayart M., Models and Languages for the Interoperability of Smart Instruments, in: *Proc. of the 2nd IFAC Symposium on Intelligent Components and Instruments for Control Applications, SICICA 94*, (Budapest June 94), Hungary, 1994, p 1-12.

[6] Foulloy L., Galichet S., Josserand J.F., Fuzzy components for fuzzy control, in: *Proc of the 2nd IFAC Symposium on Intelligent Components and Instruments for Control Applications, SICICA 94*, (Budapest June 94), Hungary, 1994.

[7] Josserand J.F., Mauris G., Benoit E., Foulloy L., Fuzzy components network for distributed intelligent systems, in: *Proc. of the International Workshop on Intelligent Robotic Systems, IRS 94*, Grenoble, july 94.

[8] Ren C. Luo, M.H. Lin, R.S. Scherp, Dynamic multisensor data fusion system for intelligent robots, in: *IEEE Journal of robotics and automation*, Vol. 4, No 4, Aug. 1988, p. 386-396.

Contact point: Eric Benoit, LAMII/CESALP, 41 av. de la Plaine, B.P. 806, F-74016 Annecy Cedex, France.
Phone +33 450 66 60 44, Fax +33 450 66 60 63, E-mail benoit@univ-savoie.fr